In the Specification:

Please amend the Specification as follows:

Please amend the paragraph beginning on page 8, line 4, as follows:

For hardware optimization, one significant potential benefit of exploiting ~~phase behavior~~ phase behavior is to optimize the usage of architecture components to phase behavior for power optimization. Optimizations have been proposed including cache re-configuration, processor width adaptation, multi-core execution, and more. Examples of ~~this~~ these are briefly summarized. Most of these optimizations try to save energy, while maintaining performance.

Please amend the paragraph beginning on page 8, line 15, as follows:

I-cache usage, for example, may show either a low number of misses or a high number for different intervals of execution. During phases with high usage, it may be better to use a larger cache, and then to shut down the sets or ways for the phases that do not need the additional storage. This can save energy while potentially maintaining performance. To address this inefficiency, phase information can be used ~~to~~ for dynamically reconfiguring caches and other structures with the intention of saving energy.

Please amend the paragraph beginning on page 8, line 22, as follows:

Another form of re-configurable cache that has been proposed dynamically divides the data cache into multiple partitions, each of which can be used for a different function, such as instruction reuse buffers~~,~~ or value predictors. These techniques can be triggered at different points in program execution including procedure boundaries and fixed intervals. The overhead of re-configuration can be quite large, so it can be important to make these policy decisions only when the large scale program behavior changes. Phase classification can be used to track this behavior to minimize overhead while guaranteeing adequate sensitivity to attain maximum benefit.

Please amend the paragraph beginning on page 10, line 3, as follows:

Phase information can potentially also be used to ~~also~~ guide adaptive compiler optimizations. An exemplary optimization is the creation of optimized code ~~``packages''~~ “packages” that are targeted towards a given phase, in order to specialize the program’s behavior to that phase of execution. This can result in faster execution, lower power, or many other run-time benefits.

Please amend the paragraph beginning on page 36, line 9, as follows:

FIG. 12 shows the effect of using only a limited amount of simulation time. As shown in the example of FIG. 12, the amount of simulation time was limited to only 300 million committed instructions, starting at the instruction, in hundreds of millions, shown in the first column of FIG. 12. As shown, the error rate has gone up over that in FIG. 11. However, because the starting point was carefully selected with the above preferred algorithms, the results in this example were within acceptable bounds. The worst case IPC difference is ~~6%~~6%.

Please amend the paragraph beginning on page 37, line 14, as follows:

Statistics for components other than hardware-independent metrics such as basic blocks are contemplated as well. As another nonlimiting example, one can try to find phase behavior by creating an IV where the components include both (1) branch instructions, and (2) load instructions. The statistic being gathered for branch instructions, for example, could be a number of branch mispredictions that occur for each individual branch instruction and the number of cache misses for each individual load instruction both gathered via hardware counters or simulation. The length of the interval in this example would be the number of branch and load instructions executed during the gathering of the interval vector. The interval vectors may then be normalized by this length and then directly compared to find their similarity. It will be understood by those in the art that various other combinations of statistics, program components, and intervals exist, and these examples should not be taken as limiting the invention to these exemplary methods.

Please amend the paragraph beginning on page 71, line 4, as follows:

The number of bits of information retained for analysis is related to the number of the buckets ~~110~~106, which is N. As the number of the buckets ~~110~~ 106 is increased, the data is spread over more buckets (table entries), making for less entries per bucket (better resolution), but at the cost of more ~~are~~ area (both in terms of number of buckets and more bits per bucket). Preferably, any distribution into the buckets ~~110~~ 106 should provide useful information. To accomplish this, it is necessary to insure that, even if data is distributed perfectly evenly over all of the buckets ~~110~~106, information is recorded about the frequency of those buckets. This can be achieved, for example, by reducing the accumulator counter by:

Please amend the paragraph beginning on page 71, line 15, as follows:

If the number of the buckets ~~110~~ 106 and interval size are powers of two, this indicates a simple shift operation. For the number of buckets ~~110~~ 106 selected in the example given (thirty-two), and the interval size profiled, this reduces the bucket size down to six bits, and preferably requires twenty-four bytes of storage for each unique phase in the Past Signature table of FIG. 28. Typically, the top six bits of the counter 112 are more than enough to distinguish between two phases, however it is possible that one or two additional bits, for example, may be needed to reduce quantization error.

Please amend the paragraph beginning on page 72, line 1, as follows:

To examine the aliasing effect described above, and to determine what an appropriate number of buckets ~~110~~ 106 should be, FIG. 30 shows the sum of the differences in the bucket weights found between all sequential intervals of execution. The y-axis shows the sum total of differences for each program. This is calculated by summing the differences between the buckets ~~110~~ 106 captured for interval i and i-1 for each interval i in the program. The x-axis is the number of distinct buckets used. All of the results are compared to the ideal case of using an infinite number of buckets ~~110~~ (or one for each separate basic block) to create the signature. On the program gcc, for example, the total sum of differences with 32 buckets was 72% of that captured with an infinite number of buckets. In general, 32 buckets has been found to be enough to distinguish between two phases.

Please amend the paragraph beginning on page 83, line 1, as follows:

Just-in-time (JIT) systems are assisted, as efficient JIT systems can be built to guide when to spend time on optimizing code. By using a hardware-independent metric for the component such as the code executed, analysis may be performed in a very short amount of time, on the order of how long it takes to execute the program itself, using a very fast high level code profiler. ~~Reoptimiztion~~ Reoptimization of a program can be expedited by determining when to perform the reoptimization.